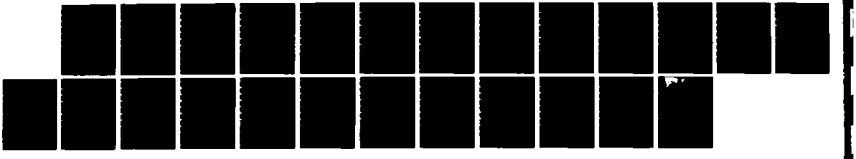
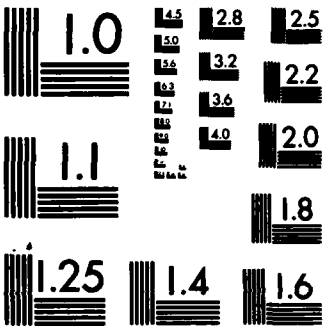
AD-A140 497 THE ROE FILE SYSTEM(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE C S ELLIS ET AL. MAR 83 TR-119 N00014-82-K-0193 1/1

UNCLASSIFIED F/G 9/2 NL





The Roe File System

Carla Schlatter Ellis and Richard A. Floyd
Computer Science Department
University of Rochester
Rochester, NY 14627

TR119
March 1983

Department of Computer Science
University of Rochester
Rochester, New York 14627

The Roe File System

Carla Schlatter Ellis and Richard A. Floyd
Computer Science Department
University of Rochester
Rochester, NY 14627

TR119
March 1983

Abstract

Roe is a network-wide file system being developed for a heterogeneous local network. The system has been designed for two purposes: to serve as a testbed for experimenting with various policies for file migration and distribution strategies and to provide users with a logically coherent file system that takes advantage of distributed and diverse resources. The system is a synthesis of solutions to the problems of ensuring consistency of replicated data, allowing transparent reconfiguration, and providing adequate file accessibility. This report describes what has been accomplished so far in the Roe project. We outline the assumed environment, the basic structure of Roe, and the functions provided. Mechanisms are presented that allow migration, replication of file objects, and replication of access information to work together. Finally, the state of the implementation to date is described.

1. Introduction

Roe is a network-wide file system being developed for a heterogeneous local network. The system has been designed for two purposes: to serve as a testbed for experimenting with various policies for file migration and distribution strategies and to provide users with a logically coherent file system that takes advantage of distributed and diverse resources. The system is a synthesis of solutions to a number of different problems such as ensuring consistency of replicated data, allowing transparent reconfiguration, and providing adequate file accessibility. The project has given us a context within which to investigate how these issues interact with each other.

While sharing of files among machines is widely recognized as one of the primary services that should be offered by a network, it has not always been convenient or easy to do. In the past, the approach has been to use the existing local file systems and provide tools for transferring files between them. This usually requires the user to know the locations of files and several different naming conventions. In an evolving network environment, one might even find different file transfer protocols depending on the source and destination machines. Given the ability to transfer files, users can spread copies of a file throughout the network in an effort to ensure availability. This puts a burden on the user to maintain consistency of these copies during subsequent updates. Our experience suggests that it is extremely easy to lose track of where all the copies are kept and what copies are up to date (even when the file is not shared). Roe grew out of a need to address these and other problems that arose in our heterogeneous local network.

The following goals have guided the design of Roe:

1) Availability. As suggested earlier, networking has encouraged users to replicate their files in order to increase availability in spite of failures. Roe supports replication of not only file objects but also the information used to find and access them. Thus the do-it-yourself management of copies has been taken over by the system. The user now deals with an abstract file object called a Roefile which is represented by a set of copies and given a symbolic name by its owner.

2) Correctness and consistency of copies. We take the point of view that the user should not be presented with any surprises (e.g. updates appearing to have disappeared right after a supposedly successful write). Therefore, consistency is given a high priority even though it implies somewhat lower availability and performance.

3) Network transparency. Our view of transparency is that the user need not be aware of the location of the data being accessed, the number of copies that may have been created, and the current status of other sites on the net. The user perceives a single directory structure for resolving the symbolic Roefile names. The names do not include any encoding of location.

4) Ability to reconfigure. It should be possible to move copies and to change the number of copies associated with a Roefile. This allows for more effective balancing of system-wide disk storage allocation and performance. Migration might be triggered by changes in usage patterns, device characteristics, or file specifications.

Although there has been a considerable amount of work done in the area of distributed file and database systems, few projects have pursued a similar set of goals. The LOCUS project at UCLA [Popek 81] seems to be the most closely related. The

emphasis there is on network transparency, availability of resources, and performance. Key elements in achieving these goals include a network-wide location independent naming structure, automatic replication of storage, a centralized synchronization scheme, automatic detection and limited resolution of inconsistencies resulting from updates during partitioned operation, and problem-oriented low level protocols. Limited migration (i.e. automatic archiving) and the ability to cache files at one's personal workstation have been included in the design of the Central and Spice file systems at CMU [Accetta 80, Thompson 80]. Distributed directory service has been provided in the Clearinghouse [Oppen 81] and Grapevine [Birrell 82]. In System R* [Lindsay 80], catalog entries for an object may be replicated, but the name of the object encodes location information for finding the proper entry. Other distributed file servers (e.g. Juniper at Xerox PARC [Sturgis 80] and SWALLOW at MIT [Svobodova 80 , Reed 80]) provide low-level functions that would allow a file system supporting name service, replication, and so on to be implemented as a client; but they do not address these issues directly.

This report describes what has been accomplished so far in the Roe project. Briefly, the heterogeneity of the native file systems has been dealt with and mechanisms have been developed that allow migration, replication of file objects, and replication of access information to work together. We have not yet started studying the performance of alternative policies built on these mechanisms. In the next section, we outline a model for local network environments that captures the underlying assumptions of this work and elucidates many of the issues being addressed. Then we outline the basic structure of Roe and the functions provided. In Section 4, algorithms for mutual consistency are evaluated with respect to our model and compatibility with migration and replication of directory information. Finally, the state of the implementation to date is described and some conclusions drawn.

2. Local Network Model

The project was conceived in terms of our local environment which is a heterogeneous set of machines (currently Xerox Alto personal computers [Thacker 79], several VAXes running UNIX™, and the RIG gateway [Ball 76, Lantz 80] running on a set of Eclipses) communicating through an Ethernet [Metcalfe 76]. The model being presented here is intended to capture those aspects of a local net that are relevant for the file system design and to be general enough to represent various types of networks, including our future plans for growth. One of the key assumptions is that with the participation of personal computers, the possibility of a number of machines being unavailable for relatively long periods of time becomes a real issue.

Figure 1 shows our network represented by the model. A *site* (denoted by a circle) is a named local file server. This definition is motivated by the existence of removable and interchangeable disks. The set of file system processes that provide access to a certain collection of data seems to be the meaningful entity to name; even though it may not permanently reside on any particular machine. Thus one of the properties associated with a site is its availability (i.e. probability of the site being 'down' because either it is off-line or has failed). Other attributes deal with the heterogeneous nature of the environment (e.g. the hardware base needed by the site). Each site can possess a number of storage devices (denoted by the symbol of a drum) with potentially different properties. For example, it is possible for some of the devices to have a higher probability of being down than its site does. An active file server could accommodate for failed devices in some respects so that the unavailable device would have a different effect than if the site were unavailable. The device has

other important properties associated with it, namely local speed of access and capacity. The network *configuration* consists of a passive *wire* (denoted by the heavy solid lines), *switches* (denoted by rectangular boxes), and addressable *machines* (denoted by heavy dots). The wire is assumed to be a high bandwidth channel which is generally reliable. The switches are used to identify places where the line of communication may be broken. A failure of a switch can cause the network to become partitioned with healthy sites that cannot communicate with each other unless an alternate path is found. The machines serve as placeholders in the configuration to which sites may be assigned (dashed lines constrain the possible mappings). *Users* may also be associated with machines from which they execute programs that use the file system.

The *status* of the model is an instantaneous description composed of the current machine-site and machine-user mappings, the state ('up' or 'down') of each site, switch, and device, and the available space left on each device. Using the status description, it is meaningful to talk, for example, about the distance from a user to a copy of the desired file (i.e. the number of switches between the machines involved). This model is reflected in the implementation by various data structures.

3. System Overview

3.1 Roe Files

As stated previously, a Roefile is an abstraction that is referred to using a location independent user-chosen symbolic name. Thus the global name space is independent of the physical configuration of the network machines and data residing on them. Names are resolved by a global directory which should be thought of as a single entity although it may have a distributed implementation. The global directory is organized as a UNIX-like hierarchical structure. A symbolic name takes the form of a path name appended with a version number. Each different version of the same pathname is a distinct object. The version number should not be confused with a timestamp which changes at each update; in other words, existing versions may be modified and new versions are created explicitly. The Roefile name maps to an object descriptor that contains the set of unique local file ids which represent copies of the object. It also contains the various attributes associated with objects (e.g. access control list, multiple copy update data, semantic type information). When a Roefile is opened, the mapping is done and an appropriate subset of copies is chosen to participate in the transaction. We have adapted Gifford's Weighted Voting algorithm [Gifford 79] for maintaining mutual consistency. This decision is discussed further in Section 4. Roe uses locking at the level of individual copies and atomic transactions to implement this solution.

3.2 Functions

This section describes the functions available to users of Roe. Communication with the file system is done through port-based IPC messages as proposed in [Rashid 80] and extended throughout our network [Moore 82].

The messages of the protocol fall into three categories represented by three types of destination ports. *Packports* provide a way of gaining access to the file system. These are public ports that are located through IPC's minimal name service. *Directoryport* messages include commands for manipulating the directory structure, changing the working directory, modifying the entries in a directory, opening files, and managing replication and migration. *Fileports* are used for operations on

individual files. A detailed description of the protocol is given in the Appendix.

A user must be logged in to the Roe system in order to issue requests. A LOGIN message containing information necessary for authentication is sent to a Packport. The reply contains a DirectoryPort to which the user sends subsequent directory operations.

DirectoryPort messages include "standard" commands such as CREATEDIR, DELETEDIR, READDIR, CHDIR, RENAME, and DELETE. The OPEN operation returns a FilePort for subsequent file access if an appropriate quorum of copies can be gathered. For a file that is being created, the message must contain the desired location of each copy and the voting configuration. In addition, the user initially has available messages to manage replication. CACHE creates a current copy of the named Roefile at the specified site (usually the user's local disk). There is also an UNCACHE command. MIGRATE moves the copy of the given Roefile that resides on the specified source pack to the destination pack.

Upon opening a file and receiving a FilePort, the user has available the ordinary file operations (e.g. READ, WRITE, CLOSE). Data sent in a WRITE operation to the FilePort is propagated to each copy participating in the transaction.

A side effect of the Roe project has been to provide uniform FTP access to non-Roe files managed by the native file systems [Bukys 82]. The messages are similar to those described in the Appendix except CACHE, CACHEDIR, MIGRATE, UNCACHE, and UNCACHEDIR do not apply and certain aspects of other operations (namely those that concern copies of a file object) are different. This protocol (with slight variations) is used internally by Roe.

3.3 Organization and Distribution

The structure of Roe is given in Figure 2. There are five types of modules shown. *Transaction Coordinators*, *Global Directory Servers*, *Local Representatives* and *Local File Servers* are considered part of Roe. *Users* are the source of commands coming into the net file system and are not actually part of it (however, a user interface [Floyd 82a] has been supplied for the Alto Mesa environment [Mitchell 79]).

The rules for distributing these modules are as follows: Each site holding copies of Roefiles has a Local File Server running on its machine and an associated Local Representative (usually co-resident). Each User is associated with a Transaction Coordinator that normally runs on the user's machine. The Global Directory is drawn as a cloud in Figure 2 since it functions as a single entity. The design allows for multiple cooperating modules delivering this service. There are no real restrictions on how these should be distributed; however in the current phase of the project, coding of the Global Directory Servers has been limited to the VAXes.

The Transaction Coordinator is responsible for transforming the User's requests for a single Roefile into a transaction involving a set of copies. It must maintain the state necessary for recovery of its transactions.

The Global Directory Subsystem does the mapping from Roefile names to the set of local file ids of the copies and selects quorums. When a user logs in, the Global Directory Server spawns a Transaction Coordinator. It also maintains the global directory structure. Migration policies will eventually be incorporated into this subsystem for automatic system-controlled migration. When a file is being created,

the location and weight of each copy to be created must be determined here. Other functions include authentication, maintaining working directories and constructing the network model.

The Local File Server is responsible for storage of files and machine dependent details. The functions provided include unique naming for the individual copies of files (<pack name><local file id>), management of disk storage space, recording of the data needed by the consistent update algorithms (e.g. timestamp, weight of the copy) and migration procedures (e.g. syntactic type), synchronization (e.g. locks), and a standard set of file manipulation operations (e.g. read, write, commit, create). At this level a file copy is treated as a stream of data units (the size and format of which depend on the intended hardware base and the type) and is locked as a whole (there is no finer granularity). The purpose of the syntactic type associated with each copy is to allow conversions between the different data representations understood by the heterogeneous computers and thus to facilitate transparent movement of data. The Local File Servers also implement the uniform FTP discussed in the previous section.

The Local Rep and Local File Server work together to realize Roefile copies. The Local Rep augments the Local File Servers to satisfy the requirements of Roe. It also establishes connections with the Global Directory Subsystem and constructs the local version of the net model.

As an example of how this works, imagine that the User in Figure 2 wants to write to an existing Roefile (only the message flow essential to understanding the relationships of the modules is shown). The LOGIN and creation of the Transaction Coordinator is assumed to have already occurred. The User issues an OPEN request (a) to the Directory Port. The Transaction Coordinator which receives this request communicates the information to the Global Directory Servers (b). The mapping is done and a quorum collected through the appropriate Local Reps (c). A set of FilePorts representing the quorum is sent back to the Transaction Coordinator (d) for use in translating incoming WRITE messages (e) to multiple updates on copies (f).

4. Replication

As mentioned earlier, Roe replicates file objects and directory information to increase availability. This requires that some algorithm be adapted to insure that copies seen by the user remain mutually consistent. In section 4.1 we describe a number of mutual consistency algorithms. Section 4.2 examines the behavior of several of these algorithms in the presence of partitioning, node failures and migration and shows why we believe that the choice made for Roe (weighted voting) is appropriate for file objects in our environment. Section 4.3 discusses the replication of directory information.

4.1 Mutual Consistency Algorithms

There are 3 basic methods of insuring an appropriate degree of mutual consistency between copies [Bernstein 82]: 'do nothing', primary copy and voting. 'Do nothing' algorithms write to all copies of a replicated file when making an update. These algorithms actually decrease the availability of a file and so will not be considered further.

Primary copy algorithms (e.g. [Stonebraker 79]) designate one copy to be the primary at each point in time. All reads and writes go to the site of this copy, and it

is responsible for notifying other copies of changes. As long the primary site is up, reads and writes may continue (one usually also requires that a majority of the sites be accessible to avoid problems with partitioning). If the primary site fails, an election is held to decide on a new primary copy. For example, in [Stonebraker 79], the primary copy depends on the status of the network (which sites are up) and a fixed linear ordering of the copies of an object. To decide who this is, each site accumulates a list of up sites and then checks to make sure that all agree on who the new primary is. This method requires that all live copies be current. This is usually handled by postulating an underlying message transmission system that can buffer an arbitrary number of messages for later transmission to a down site.

An alternative primary copy algorithm, used in LOCUS [Popek 81], employs a centralized coordinator to keep track of the state of each copy of a file. Requests to open a file go through the centralized coordinator, which returns a pointer to the most current version of the file. After the file is closed, the coordinator propagates changes to the other copies of the file. The scheme used in LOCUS does not ensure consistency. If the network becomes partitioned and separates the coordinator from some of its files or if the site of the coordinator goes down, a new coordinator is created. This may lead to loss of currency information or more than one coordinator controlling access to a group of files, which can result in incompatible versions of a file being created. Loss of the primary before updates can be propagated to other copies can cause similar problems. The centralized coordinator approach can be preserved in a consistent algorithm. With the consistency requirement, solutions that can be classified as primary copy appear to share two important characteristics: the need for *agreement* on a single authority governing the object (e.g. the identity of the coordinator or the primary copy itself) and a method for propagating currency information to candidates that may take over the primary role.

Voting solutions do not require that a copy be brought up to date when a site recovers since the existence of obsolete copies is acceptable as long as enough current copies are available. Weighted Voting [Gifford 79] associates a timestamp and a voting strength with each copy of a file. When a file is opened, the timestamp and voting strength are collected from the copies. At least r votes (a read quorum) must be collected to read a file and MAX[r,w] to write it. Reads can be from any current copy and writes go to current copies which hold a total of at least w votes (a write quorum). Making $r + w$ greater than the total number of votes in all copies of the file insures that at least one current copy will be in any quorum. The timestamp of each participating copy is incremented when the copy is updated.

4.2 Evaluation of Mutual Consistency Algorithms

These algorithms can be evaluated along two dimensions important for Roe; namely, what complications are introduced by the migration requirement and what the perceived delay is to the user due to coordinating updates. Comparisons of delay are made by considering each algorithm in an identical context that consists of the user, local file servers, and the directory manager. Note that this is simpler than the actual organization of Roe described in section 3.3. Performance figures given include the overhead for coordinating updates and maintaining currency information, but not for directory lookup. See [Floyd 82b] for details of the analysis. In addition, the algorithm chosen should be easy to implement within the constraints imposed by our network model.

[Stonebraker 79] and similar algorithms provide the strong consistency desired for Roe. However, one of the assumptions made to insure this is unbounded message

queues for down sites. Since some Roe sites reside on personal machines and so may be down for long periods of time, this is unacceptable. Along with the issue of queue length, queuing of updates implies a recovery time whenever a site is brought up. Because of the direct interaction between users and their personal machines, the delay of bringing the local disk up to date will seem significant. The delay associated with opeining a file depends on the status of the primary copy. If it is down, an election is triggered which is a very expensive operation in terms of message activity. Otherwise, opens are simple, involving a constant number of messages (3 under our assumptions). Writes may be propagated in parallel to all copies. Thus in this case, the delay is 3d (d = message delay). The requirement that each server holding a copy know the locations of all other copies makes migration difficult, since all copies have to be informed when one moves. Each list of copies *must agree* in order for the determination of the primary copy to work. Creating new copies (e.g., caching a temporary copy to increase performance) also requires that all other copies be informed. Queuing of updates also interacts with migration. A reasonable restriction to make is that a copy may move when its queue is empty and the file is not opened for writing.

As explained in section 4.1, the solution in [Popek 81] fails to meet our consistency objective. A compensation for the poor consistency is that availability is very high. Migration and caching are both easy, since only the centralized coordinator needs to be notified when copies are moved or created. Performance is high. Communication and maintenance of currency information takes as few as three messages and two disk accesses to open a file for write, with a time delay (since updating currency information can be done asynchronously) of roughly 3d + a (a = disk access). Also, with this solution, writes may be directed to the fastest of the sites holding copies of a file and distributed to slower sites at a later time. [Gifford 79] and [Stonebraker 79] generally require writes to proceed at the speed of the slowest participating site.

[Gifford 79] also provides the strong consistency desired for Roe. Availability is normally comparable to [Stonebraker 79], although the weights of votes on copies can be adjusted based on knowledge about site availability to increase file availability. Migration and caching involve just updating directory entries. Attempting to collect a quorum based on a not yet updated directory entry is not a problem. A copy that is 'in transit' simply does not vote. If a write quorum can be gathered from other representatives, the moved copy could be outdated when it is installed in the new site but such inaccuracies can be tolerated. [Gifford 79] lends itself to distribution more readily than [Popek 81], since currency information for a copy is kept with the copy itself and is always correct. One pays for this in the complexity of file opens. Opening for write takes at least (2+2n) messages and 2n disk accesses, where n is the number of copies participating in the vote. (The example in Figure 2 actually takes somewhat more than this because of the presence of the Transaction Coordinator, Local Reps and because votes are being collected in the Global Directory cloud.) For our purposes, n will usually be a small number (e.g., three). Since vote collecting can go on in parallel and timestamps may be updated asynchronously, the time delay is roughly 4d + a (plus queuing delays). This can be reduced to 3d + a if the user is responsible for collecting the quorum and broadcasting updates. Although the initial activity is fairly high, the delay perceived by the user is comparable to the primary copy algorithms.

Of the algorithms outlined above, [Gifford 79] is the only one which gives the desired degree of consistency and availability while still being workable in the presence of migration. A number of other algorithms besides the ones presented here

have been examined. All either have inferior consistency or availability properties when compared to [Gifford 79] or complicate migration.

4.3 Replicating the Directory

From the list of directory operations given in section 3, it is clear that a directory server must support the following operations: Read Entry (where 'Entry' is the information describing one Roefile), Add Entry, Delete Entry, Modify Entry and Enumerate Entries at a node of the directory tree. The 'enumerate' operation and the common practice of grouping related files in a directory suggests that the unit of replication and migration be the set of entries in a node. The strong consistency requirements given for file objects also apply here (we would like to see the effects of adding a file when we subsequently do an enumerate, the effects of renames when we do lookups, and so on). Also, we will want to migrate directory information. Hence, weighted voting is a reasonable starting point from which to develop an algorithm for maintaining mutual consistency in replicated directories.

The fairly specialized entry-oriented nature of operations on a directory means that it is not necessary to lock the entire file describing a node. Locking is also not desirable since a user might want to remain 'connected' to a directory for a relatively long period of time without affecting the ability of others to access it. Based on this, we can propose the following variation of Gifford's Weighted Voting algorithm to increase concurrency:

When a user connects to a directory, the weighted voting algorithm is performed on the node copies to collect a read quorum, r. At this time, the user is *registered* with the directory servers controlling copies (registration differs from holding a lock in that access may be 'broken' with notification as explained below and the modification of individual entries is reported to connected users). From the read quorum, a current copy is selected and read requests are directed to it.

Writing requires that updates be made atomically to current copies containing at least w votes. This is done by sending the participating servers the update requests (Add Entry, Delete Entry or Modify Entry). If they are willing to make the change, they respond with an acknowledgement. If at least w votes are collected this way, then the user can instruct the servers to commit. At this point, the changes are actually made and the timestamps of the participating copies incremented. If the user is unable to collect w votes, then the request is aborted. Only one write may be active for a directory node at a time. Note that a writer needs to have collected a read quorum sometime in the past so that he can determine which are the current copies.

The Modify Entry operation needs a bit of special handling to guard against making changes based on invalid data. We send, in the Modify Entry request, both the new information and the data in the entry that the changes are based upon. If this data does not agree with the information currently in the directory, the request is rejected. Since an individual directory entry is small, little extra overhead is involved in doing this.

The registration information is used when a writer finds that he can't update all current copies of a node, even though he is able to collect a write quorum. In this case, the writer, in the commit message, tells the participating directory servers to notify users who have registered for that directory that they may no longer be reading a current copy. Since r+w is greater than the total number of available votes, there is always some overlap between read and write quorums and so all

readers will be notified.

Each copy of a directory node contains a timestamp, voting strength and an entry for each file and directory that is a direct descendent of the node. Each entry contains the location of every copy of the file (in the form [pack]<unique id>) described by the entry, the read and write quorums for the file, and access control and other miscellaneous information (Figure 3).

When a user first logs in, a Global Directory server is contacted which locates his default directory and registers him with the appropriate directory servers. This may require traversing down one or more nodes to reach the user's directory. Information about this path is cached so that the user can easily connect to ancestors of his default directory (and to speed up future logins).

5. Concluding Remarks

This paper has described Roe, a file system designed for a heterogeneous local network. The principal goals have been file availability (achieved through replication), consistency, network transparency, and the ability to migrate data.

At this time, Roe is partially implemented. One outcome so far has been to deal with the heterogeneity of our environment. This has involved defining a common notion of 'file copy' that can be realized from various native file servers and that is useful for the functions being supported by Roe. As mentioned before, completion of the Local File Servers has also provided a uniform FTP for access to non-Roe files (named by location and local name in a uniform syntax). Most site-dependent issues are taken care of within this operational part of Roe and so the remainder of the system sees a uniform interface. Another side effect of this effort has been a number of improvements made to the network IPC motivated by the needs of Roe.

Specification of the Global Directory Servers and the Transaction Coordinator has convinced us that it is possible to integrate migration, file replication, and replication of directory information. This work has led to insights into the desirability of solutions that can tolerate certain inaccuracies in the data used for management. It has also helped develop ideas about distributed job management described in [Ellis 82]. We expect a VAX-based directory server and a prototype Transaction Coordinator to be implemented by the end of summer 1983.

After completion of the first stage of the project, we plan to investigate file migration policies, an area in which relatively little is known. Our intention is that this file system implementation will serve as a vehicle for experimentation with a number of different policies. Candidates include some approximation to Least Recently Used for removing copies from sites of high demand to long term storage, creating a temporary cached copy on demand, pre-fetching or updating a locally cached copy in anticipation of use based on the user's behavior in recent sessions or relationships between a file already requested and other files, and upgrading a copy to a more desirable site when recent demand for the file has increased above some threshold. These involve monitoring file usage patterns on a per-user basis (e.g. the user's login profile might include a list of files used in previous sessions) and on a per-site basis. Other measurements will be collected for the purpose of evaluating policies. Migration may be triggered by specific events such as the creation of a new file causing demand for more storage, an explicit command to change the properties associated with a file, an observed increase in network traffic, or a user logging in. The overhead involved in doing conversion between representations may influence

the decision of whether or not to move a particular copy.

Another issue we plan to consider is the distribution of directory information. The design allows for replication within the directory structure at a relatively fine granularity. Thus it will be possible to try out various strategies for partitioning the structure. For example, the unit of distribution for a particular policy might be a subtree of the hierarchy and an appropriate location for it might be at the owner's site. Alternatively, it may be desirable for a directory to be placed at a site that contains a large amount of the data it refers to.

Future possibilities for research related to this project include software development tools and various aspects of distributed job management.

Acknowledgements

The preparation of this paper was supported in part by the National Science Foundation under Grants IST−8025761 and MCS−8104008 and by the Defense Advanced Research Projects Agency under N00014−82−K−0193.

6. Bibliography

[Accetta 80] M. Accetta, G. Robertson, M. Satyanarayanan and M. Thompson "The Design of a Network-Based Central File System," Technical Report CMU-CS-80-134, Carnegie-Mellon University, August 1980.

[Ball 76] J.E. Ball, J.A. Feldman, J.R. Low, R.F. Rashid, and P.D. Rovner "RIG: Rochester's Intelligent Gateway System Overview," *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, December 1976, 321-328.

[Bernstein 82] P. Bernstein and N. Goodman "A Sophisticates Introduction to Distributed Database Concurrency Control," TR-19-82, Aiken Lab, Harvard University, 1982.

[Birrell 82] A. Birrell, R. Levin, R. Needham, and M. Schroeder "Grapevine: An Exercise in Distributed Computing," *CACM 25*:4, April 1982, 260-274.

[Bukys 82] L. Bukys and R. Floyd "Even More FTP Cogitation," Internal Document, Computer Science Dept., Univ. of Rochester, June 1982.

[Ellis 82] C.S. Ellis, J.A. Feldman, and J.E. Heliotis, "Language Constructs and Support Systems for Distributed Computing," *Proceedings*, ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, Ottawa, Canada, August 1982.

[Floyd 82a] R. Floyd "Notes on the Roe User Interface," Internal Document, Computer Science Dept., Univ. of Rochester, February 1982.

[Floyd 82b] R. Floyd
"Mutual Consistency Algorithms for Roe,"
Internal Document, Computer Science Dept., Univ. of Rochester, December 1982.

[Gifford 79] D. Gifford
"Weighted Voting for Replicated Data,"
Proceedings, 7th Symposium on OS Principles, December 1979.

[Lantz 80] K.A. Lantz
"Uniform Interfaces for Distributed Systems,"
TR63, Computer Science Dept. University of Rochester, May 1980.

[Lindsay 80] B. Lindsay
"Object Naming and Catalog Management for a Distributed Database Manager,"
IBM Research Report RJ 2914, San Jose, Calif., August 1980.

[Metcalfe 76] R.M. Metcalfe and D.R. Boggs
"Ethernet: Distributed Packet Switching for a Local Computer Network,"
Communications of the ACM, 1, (7), July 1976.

[Mitchell 79] J.G. Mitchell, W. Maybury and R. Sweet
"Mesa language manual (version 5),"
TR CSL-79-3, Xerox Palo Alto Research Center, 1979.

[Moore 82] L. Moore, L. Bukys, and J. Heliotis
"Design and Implementation of a Local Network Message Passing Protocol"
Presented at the 7th Conf. on Local Computer Networks, Minneapolis, October 1982.

[Oppen 81] D. Oppen and Y. Dalal
"The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment,"
OPD-T8103, Xerox Office Products Division, October 1981.

[Popek 81] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel
"LOCUS: A Network Transparent, High Reliability Distributed System,"
Proceedings, 8th Symposium on OS Principles, December 1981.

[Rashid 80] R.F. Rashid
"An Inter-Process Communication Facility for UNIX,"
TR CMU-CS-80-124, Dept. of Computer Science, Carnegie-Mellon University, March 1980.

[Reed 80] D. Reed and L. Svobodova
"SWALLOW: A Distributed Data Storage System for a Local Network,"
International Workshop on Local Networks, Zurich, Switzerland, August 1980.

[Stonebraker 79] M. Stonebraker
"Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES,"
IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May 1979.

[Sturgis 80] H. Sturgis, J. Mitchell, and J. Israel
"Issues in the Design and Use of a Distributed File System,"
SIGOPS *Operating Systems Review*, July 1980.

[Svobodova 80] L. Svobodova
"Management of Object Histories in the SWALLOW Repository,"
MIT/LCS/TR-243, Lab. for Computer Science, M.I.T., July 1980.

[Thacker 79] C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sproull, and D.R. Boggs
"Alto: A Personal Computer,"
TR CSL-79-11, Xerox Palo Alto Research Center, August 1979.

[Thompson 80] M. Thompson, G. Robertson, M. Satyanarayanan, and M. Accetta
"Spice File System,"
Dept. of Computer Science, Carnegie-Mellon University, September 1980.

Appendix

In the following description of messages, only successful outcomes are considered. The format is OPERATION (arguments) → data in reply.

LOGIN (username, password) → DirectoryPort

A user must be logged in to the Roe system in order to issue requests. A LOGIN message containing information necessary for authentication is sent to a Packport. The server checks the validity of the user and establishes the user's working directory. The reply contains a Directory Port to which the user sends subsequent directory operations.

The following messages are sent to Directory Ports:

CACHE (Roefilename, site) → success

This message results in a current copy of the named Roefile being created at the specified site (usually the user's local disk).

CACHEDIR (path, site) → success

The path is a prefix of a Roefile name designating a directory node. This operation creates a copy of the directory on the specified site.

CHDIR (path) → success

In response to this message, the server changes the current working directory associated with the Directory Port to which the message was sent.

CHVOTES (Roefilename, property List Changes) → number of properties changed

This operation is used for modifying the voting configuration of the specified file according to the list of changes given as an argument. Based on the changes to be made, an appropriate quorum must be collected.

CREATEDIR (path [, properties]) → success

This operation creates a directory node with the specified path name. The properties stipulate the distribution of copies of this new directory.

DELETE (Roefile name) → success

This operation deletes the file.

DELETEDIR (path) → success

This operation deletes a directory node if it is empty.

LOGOUT ( ) → Success

This ends the server-client relationship and deallocates the Directory Port.

MIGRATE (Roefilename, source site, destination site) → success

This command moves the copy of the given Roefile that resides on the specified source pack to the destination pack.

OPEN (access, Roefilename, [,recovery port name] [,properties]) → FilePort

This is how users acquire FilePorts. The access string describes what will be done with the file and may be composed of the following characters:

'R' indicates that the file is to be opened for reading.

'W' indicates that the file is to be opened for writing.

'A' indicates that the file is to be opened with atomic access. Operations on the file must be (two-phase) committed before the effects become permanent. The optional argument, recovery port name, is needed for atomic access. It is used in recovering from a crash that occurs between phases of the commit protocol.

'C' indicates that the file should be created if necessary. If the file already exists, it will be truncated to zero length when the open occurs. The optional argument, properties, is used for creating a Roefile that does not already exist. Currently, properties consists of the desired location of each copy and the voting configuration.

This operation is successful if an appropriate quorum of copies can be gathered to participate in the forthcoming transaction.

READDIR (destination port [, pattern]) → number of items read

This operation is used to obtain a list of files in the working directory, optionally filtered through a pattern matcher first. The file names are sent to the indicated port in a WRITE operation.

READROEPROPERTIES (Roefilename [, property name]) → property List

This message requests that directory-level properties associated with the named file be returned. Optionally, only a selected properties (e.g. locations of copies) can be requested.

REGISTER (destination port)

This operation allows the client to maintain an up-to-date idea of the contents of a directory. Status changes for files in the directory result in file names being sent to the specified port in the same format as the results of a READDIR.

RENAME (old Roefilename, new Roefilename) → success

The file object reached by the old Roefilename is made accessible by the new Roefilename and no longer accessible by the old name.

UNCACHE (Roefilename, site) → success

and

UNCACHEDIR (path, site) → success

These remove the cached copy from the given site.

The remaining messages are sent to File Ports

ABORT ( ) → success

This is used only if the file was opened with Atomic access. It undoes all operations subsequent to the previous Commit point.

CLOSE ( ) → success

This message ends access to the resources associated with the File Port and deallocates it. If the file was opened with atomic access, any uncommitted operations are aborted.

COMMIT ( ) → success

This is used only if the file was opened with Atomic access. It is the second part of the two-phase Commit protocol.

READ (destination port, number Of Items To Read, number of items per block) → number Of Items READ

This message requests that some number of items from the file to be written (using the FilePort WRITE operation) to the specified destination Port. Reading begins at the current position of the read-write pointer; when the operation is done the read-write pointer is positioned after the last datum read.

READPROPERTIES ([propertyName]) → property List

This message requests that the file's property list be returned. A selected property may be read by including the optional propertyName argument.

SEEK (position) → success

This message changes the value of the read-write pointer.

SYNC ( ) → success

This is used only if the file was opened with atomic access. It is the first part of the two-phase commit protocol. A success reply is interpreted as willingness to commit.

TELL ( ) → position

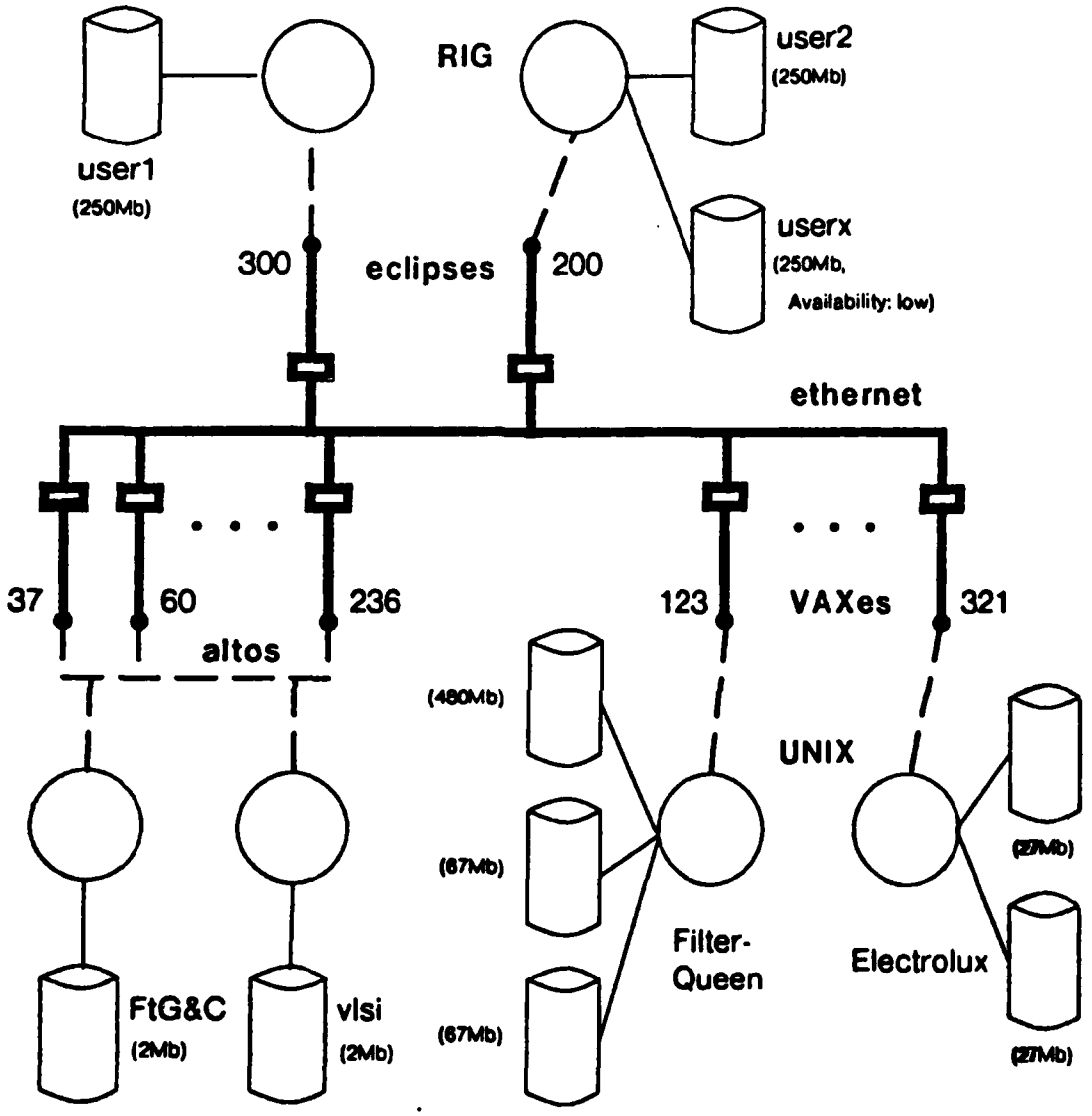
This operation returns the value of the read-write pointer.

{ WRITE (data) }* WRITEACK (data) → number Of Items written

The WRITE operation is special in that the request is composed of multiple messages. The result returned is the total number of items (from the preceding stream of WRITE messages and the final WRITEACK message) which were successfully written. Data sent in a WRITE operation to a File Port is propagated to each copy participating in the transcation.

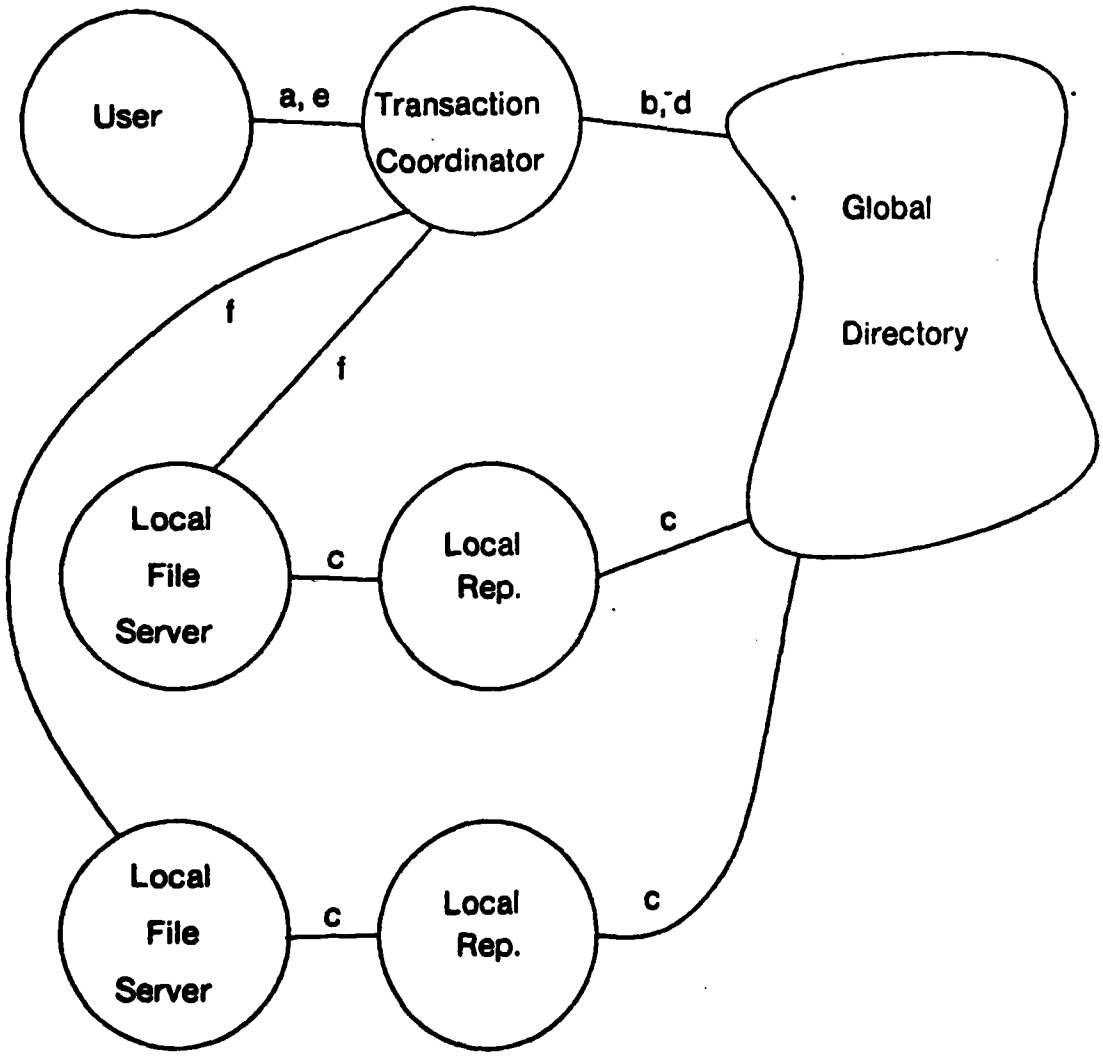
WRITEPROPERTIES (property List Changes) → number of Properties Changed

This operation modifies the file's properties (other than location of copies and voting configuration) according to the argument given.



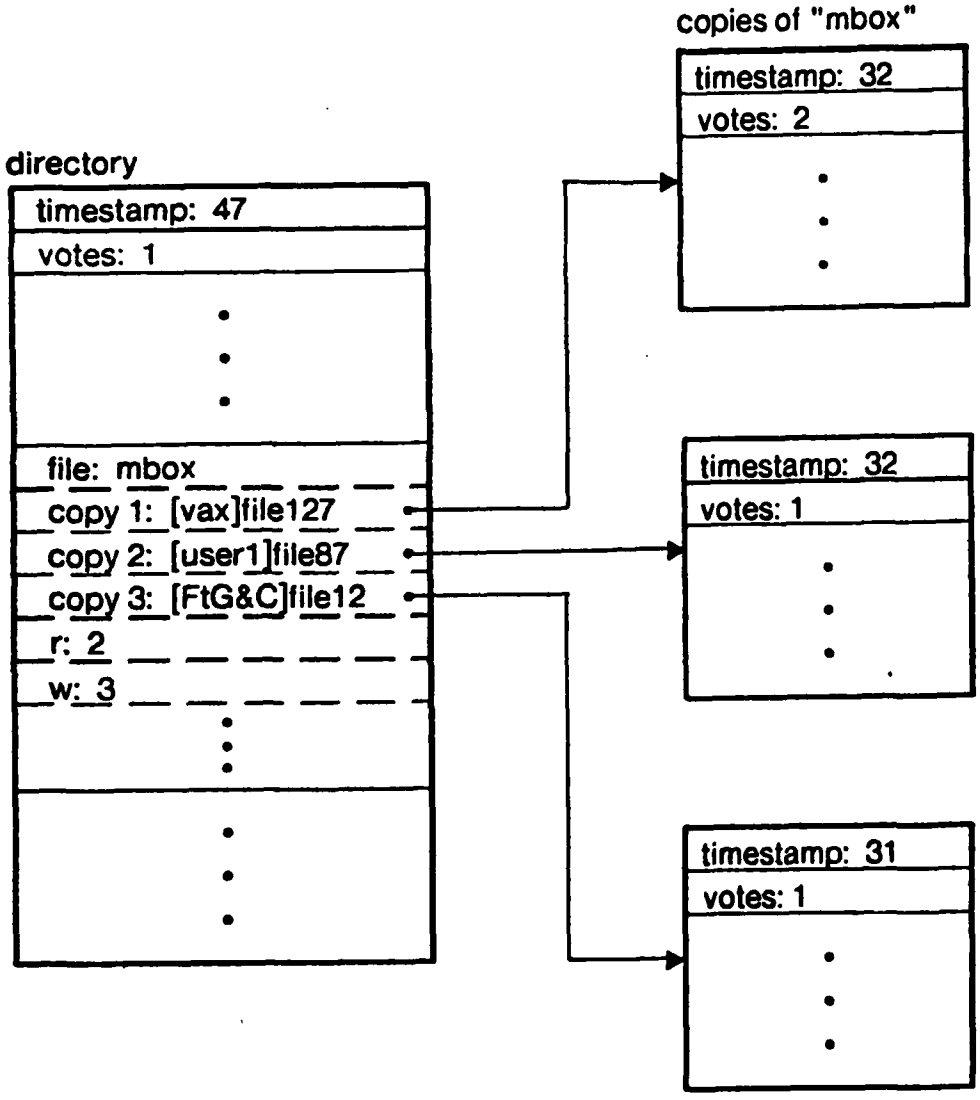
U of R Local Network

Figure 1



Overall Roe Structure

Figure 2



A Sample Directory

Figure 3

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: TR119	2. GOVT ACCESSION NO. A140497
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: The ROE File System	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Carla Schlatter Ellis and Richard A. Floyd	8. CONTRACT OR GRANT NUMBER(s): N00014-82-K-0193
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, University of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, Virginia 22209	12. REPORT DATE: March 1983
	13. NUMBER OF PAGES: 19
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research, Information Systems, Arlington, Virginia 22217	15. SECURITY CLASS. *(of this report)*: unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Network file system; distributed systems; consistency; reliability; migration; transparency; replication.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

ROE is a network-wide file system being developed for a heterogeneous local network. The system has been designed for two purposes: to serve as a testbed for experimenting with various policies for file migration and distribution strategies and to provide users with a logically coherent file system that takes advantage of distributed and diverse resources. The system is a synthesis of solutions to the problems of ensuring consistency of replicated data, allowing transparent reconfiguration, and providing adequate

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

file accessibility. This report describes what has been accomplished so far in the ROE project. We outline the assumed environment, the basic structure of ROE, and the functions provided. Mechanisms are presented that allow migration, replication of file objects, and replication of access information to work together. Finally, the state of the implementaiton to date is described.